应用程序API

Applications API

single-spa exports named functions and variables rather than a single default export. This means importing must happen in one of two ways:

import { registerApplication, start } from 'single-spa';

// or

import \* as singleSpa from 'single-spa';

single-spa输出的是命名函数和变量而不是默认输出，这意味着引用必须用以下两种方式：

import { registerApplication, start } from 'single-spa';

// 或

import \* as singleSpa from 'single-spa';

#### 常用函数

##### 注册应用函数

[#](https://single-spa.js.org/docs/api#registerapplication)registerApplication

singleSpa.registerApplication(

'appName',

() => System.import('appName'),

location => location.pathname.startsWith('appName')

)

`registerApplication` is the most important api your root config will use. Use this function to register any application within single-spa.

Note that if an application is registered from within another application, that no hierarchy will be maintained between the applications.

It is described in detail inside of the [Configuration docs](https://single-spa.js.org/docs/configuration#registering-applications)

`registerApplication`是基础配置会用到的最重要的API，调用这个方法可以在single-spa中注册一个应用。

请注意，如果一个应用是从另一个应用中注册的，则不会在在多个应用之间维护层次结构。

详细解析请见 [配置文档](https://zh-hans.single-spa.js.org/docs/configuration#registering-applications)

###### arguments

arguments

appName: string

App name that single-spa will register and reference this application with, and will be labelled with in dev tools.

applicationOrLoadingFn: () => <Function | Promise>

Must be a loading function that either returns the resolved application or a promise.

activityFn: (location) => boolean

Must be a pure function. The function is called with `window.location` as the first argument and should return a truthy value whenever the application should be active.

customProps?: Object = {}

Will be passed to the application during each lifecycle method.returns `undefined`

appName: string

应用的名字将会在single-spa中注册和引用, 并在开发工具中标记。applicationOrLoadingFn: () => <Function | Promise>

必须是一个加载函数，返回一个应用或者一个Promise。

activityFn: (location) => boolean

必须是个纯函数, 该函数由`window.location`作为第一个参数被调用, 当应用应该被激活时它应该返回一个真值。

customProps?: Object = {}

在生命周期钩子函数执行时会被作为参数传入

###### returns

`undefined`

##### 启动函数

[#](https://single-spa.js.org/docs/api#start)start

singleSpa.start();

// Optionally, you can provide configuration

singleSpa.start({

urlRerouteOnly: true

});

Must be called by your single spa config. Before `start`  is called, applications will be loaded, but will never be bootstrapped, mounted or unmounted. The reason for `start`

 is to give you control over the performance of your single page application. For example, you may want to declare registered applications immediately (to start downloading the code for the active ones), but not actually mount the registered applications until an initial AJAX request (maybe to get information about the logged in user) has been completed. In that case, the best performance is achieved by calling `registerApplication` immediately, but calling `start` after the AJAX request is completed.

必须在你single spa的配置中调用！在调用 `start`之前, 应用会被加载, 但不会初始化，挂载或卸载。 `start` 的原因是让你更好的控制你单页应用的性能。举个栗子，你想立即声明已经注册过的应用（开始下载那些激活应用的代码），但是实际上直到初始化AJAX（或许去获取用户的登录信息）请求完成之前不会挂载它们 。 在这个例子里，立马调用 `registerApplication` 方法，完成AJAX后再去调用 `start`方法会获得最佳性能。

###### arguments

The `start(opts)` api optionally accepts a single `opts` object, with the following properties. If the opts object is omitted, all defaults will be applied.

`urlRerouteOnly`: A boolean that defaults to false. If set to true, calls to `history.pushState()`  and `history.replaceState()`will not trigger a single-spa reroute unless the client side route was changed. Setting this to true can be better for performance in some situations. For more information, read [original issue](https://github.com/single-spa/single-spa/issues/484).

' start(opts) ' api可选地接受单个' opts '对象，具有以下属性。如果省略opts对象，则应用所有默认值。' urlRerouteOnly ':默认为false的布尔值。如果设置为true，则调用“history.pushState()”和“history.replaceState()”将不会触发单一spa重新路由，除非客户端路由被更改。在某些情况下，将此设置为true可以更好地提高性能。有关更多信息，请[阅读issues](https://github.com/single-spa/single-spa/issues/484)。

###### returns

`undefined`

##### 应用切换时调用的函数

[#](https://single-spa.js.org/docs/api#triggerappchange)triggerAppChange

singleSpa.triggerAppChange();

Returns a Promise that will resolve/reject when all apps have mounted/unmounted. This was built for testing single-spa and is likely not needed in a production application.

返回一个Promise对象，当所有应用挂载/卸载时它执行 resolve/reject 方法，它一般被用来测试single-spa，在生产环境可能不需要。

###### arguments

none

###### returns

Promise：Returns a Promise that will resolve/reject when all apps have mounted.

返回一个Promise对象，当所有应用挂载/卸载时它执行 resolve/reject 方法。

##### 切换应用路由调用的函数

[#](https://single-spa.js.org/docs/api#navigatetourl)navigateToUrl

// Three ways of using navigateToUrl

//切换导航路由的3中方法

singleSpa.navigateToUrl("/new-url");

singleSpa.navigateToUrl(document.querySelector('a'));

document.querySelector('a').addEventListener(singleSpa.navigateToUrl);

<!-- A fourth way to use navigateToUrl, this one inside of your HTML -->

<!-- 第四种使用navigateToUrl的方法是在HTML中 -->

<a href="/new-url" onclick="singleSpaNavigate()">My link</a>

Use this utility function to easily perform url navigation between registered applications without needing to deal with `event.preventDefault()`, `pushState`, `triggerAppChange()`, etc.

使用这个通用方法来轻松的实现在不同注册应用之前的切换，而不必需要处理  `event.preventDefault()`, `pushState`, `triggerAppChange()`, 等等。

###### arguments

* navigationObj: string | context | DOMEvent

navigationObj must be one of the following types:

1、a url string.

2、a context / thisArg that has an `href` property; useful for calling `singleSpaNavigate.call(anchorElement)` with a reference to the anchor element or other context.

3、a DOMEvent object for a click event on a DOMElement that has an `href` attribute; ideal for the `<a onclick="singleSpaNavigate"></a>` use case.

navigationObj必须是以下类型之一:

1、url字符串。

2、具有' href '属性的上下文或参数;用于调用“singleSpaNavigate.call(anchorElement)”，并引用锚元素或其他上下文。

3、一个DOMEvent对象，用于具有' href '属性的DOMElement上的单击事件;适用于“<a onclick="singleSpaNavigate"></a> '用例。

###### returns

`undefined`

##### 获取已挂载应用的函数

[#](https://single-spa.js.org/docs/api#getmountedapps)getMountedApps

const mountedAppNames = singleSpa.getMountedApps();

console.log(mountedAppNames); // ['app1', 'app2', 'navbar']

###### arguments

none

###### returns

appNames: string[]

Each string is the name of a registered application that is currently `MOUNTED`

每个字符串都是当前“挂载”的已注册应用程序的名称

##### 获取应用名称的函数

[#](https://single-spa.js.org/docs/api#getappnames)getAppNames

const appNames = singleSpa.getAppNames();

console.log(appNames); // ['app1', 'app2', 'app3', 'navbar']

###### arguments

none

###### returns

appNames: string[]

Each string is the name of a registered application regardless of app status.

每个字符串都是已注册应用程序的名称，而不管应用程序的状态如何。

##### 获取应用当前状态的函数

[#](https://single-spa.js.org/docs/api#getappstatus)getAppStatus

const status = singleSpa.getAppStatus('app1');

console.log(status); // one of many statuses (see list below). e.g. MOUNTED

//许多状态之一(见下面的列表),如:MOUNTED

###### arguments

appName: string

Registered application name.

已注册应用的名称

###### returns

appStatus: <string | null>

Will be one of the following string constants, or `null`  if the app does not exist.

NOT\_LOADEDapp has been registered with single-spa but has not yet been loaded.

LOADING\_SOURCE\_CODE 's source code is being fetched.

NOT\_BOOTSTRAPPED app has been loaded but not bootstrapped.

BOOTSTRAPPINGthe `bootstrap`lifecycle function has been called but has not finished.

NOT\_MOUNTED app has been loaded and bootstrapped but not yet mounted.

MOUNTING app is being mounted but not finished.

MOUNTED app is currently active and is mounted to the DOM.

UNMOUNTING app is currently being unmounted but has not finished.

UNLOADING app is currently being unloaded but has not finished.

SKIP\_BECAUSE\_BROKEN app threw an error during load, bootstrap, mount, or unmount and has been siloed because it is misbehaving and has been skipped. Other apps will continue on normally.

LOAD\_ERROR The app's loading function returned a promise that rejected. This is often due to a network error attempting to download the javascript bundle for the application. Single-spa will retry loading the application after the user navigates away from the current route and then comes back to it.

**Note about LOAD\_ERROR status**

Note that if you're using SystemJS to load your bundles, you need to add the following code to get SystemJS to re-attempt the network request when your loading function calls `System.import()` on an application in `LOAD\_ERROR` status.

singleSpa.addErrorHandler(err => {

if (singleSpa.getAppStatus(err.appOrParcelName) === singleSpa.LOAD\_ERROR) {

System.delete(System.resolve(err.appOrParcelName));

}

})

将是下列字符串常量之一，如果应用程序不存在，则为“null”。

NOT\_LOADEDapp 已在single-spa注册，但尚未加载。

LOADING\_SOURCE\_CODE 源代码正在获取中。

NOT\_BOOTSTRAPPED app已加载但未初始化。

BOOTSTRAPPING 已调用“初始化”生命周期函数，但尚未完成。

NOT\_MOUNTED app已经加载和初始化，但还没有挂载。

MOUNTING 正在挂载app，但尚未完成。

MOUNTED app当前处于激活状态，并被挂载到DOM中。

UNMOUNTING app目前正在卸载中，但尚未完成。

UNLOADING app目前正在移除，但尚未完成。

SKIP\_BECAUSE\_BROKEN app在加载、初始化、挂载或卸载过程中抛出错误，并且由于行为不当而被跳过和被隔离。其他应用程序将正常运行。

LOAD\_ERROR 应用程序的加载功能返回了一个被拒绝的承诺。这通常是由于试图下载应用程序的javascript包时出现网络错误造成的。Single-spa将在用户从当前路由导航并返回到当前路由后重试加载应用程序。

**注意LOAD\_ERROR状态**

请注意，如果使用SystemJS加载包，则需要添加以下代码，以使SystemJS在加载函数调用' System.import() '时在' LOAD\_ERROR '状态下重新尝试网络请求。

singleSpa.addErrorHandler(err => {

if (singleSpa.getAppStatus(err.appOrParcelName) === singleSpa.LOAD\_ERROR) {

System.delete(System.resolve(err.appOrParcelName));

}

})

##### 应用移除函数

[#](https://single-spa.js.org/docs/api#unloadapplication)unloadApplication

// Unload the application right now, without waiting for it to naturally unmount.

// 立即卸载应用程序，而不是等待它自然卸载

singleSpa.unloadApplication('app1');

// Unload the application only after it naturally unmounts due to a route change.

// 只有在应用程序由于路由更改而自然卸载之后才能卸载该应用程序

singleSpa.unloadApplication('app1', {waitForUnmount: true});

The purpose of unloading a registered application is to set it back to to a NOT\_LOADED status, which means that it will be re-bootstrapped the next time it needs to mount. The main use-case for this was to allow for the hot-reloading of entire registered applications, but `unloadApplication` can be useful whenever you want to re-bootstrap your application.

Single-spa performs the following steps when unloadApplication is called.

1. Call the [unload lifecyle](https://single-spa.js.org/docs/api#unload) on the registered application that is being unloaded.
2. Set the app status to NOT\_LOADED
3. Trigger a reroute, during which single-spa will potentially mount the application that was just unloaded.

Because a registered application might be mounted when `unloadApplication`  is called, you can specify whether you want to immediately unload or if you want to wait until the application is no longer mounted. This is done with the `waitForUnmount` option.

当调用 `unloadApplication` 时，Single-spa执行以下步骤。

1. 在一个已经注册的应用上，调用 [unload lifecyle](https://zh-hans.single-spa.js.org/docs/api#unload) 方法。
2. 将次应用的状态置为 NOT\_LOADED
3. 触发路由重定向，在此期间single-spa可能会挂载刚刚卸载的应用程序。

因为在调用 `unloadApplication` 时可能会挂载已注册的应用，所以可以指定是要立即卸载还是要等到应用不再挂载。这是通过 `waitForUnmount` 参数完成的。

###### arguments

appName: string

Registered application name.

options?: {waitForUnmount: boolean = false}

The options must be an object that has a `waitForUnmount` property. When `waitForUnmount` is `false`, single-spa immediately unloads the specified registered application even if the app is currently mounted. If it is `true`

, single-spa will unload the registered application as soon as it is safe to do so (when the app status is not `MOUNTED`).

appName: string

注册应用的名字

options?: {waitForUnmount: boolean = false}

参数必是一个包含`waitForUnmount`属性的对象。当 `waitForUnmount` 是 `false`, single-spa 立刻移除特定应用，尽管它已经被挂载。 当它是 `true`时, single-spa 会等待到它的状态不再是`MOUNTED`时才移除应用

###### returns

Promise

This promise will be resolved when the registered application has been successfully removed.

当应用被成功移除时，Promise对象会被resolved。

##### 检查激活状态函数

[#](https://single-spa.js.org/docs/api#checkactivityfunctions)checkActivityFunctions

const appsThatShouldBeActive = singleSpa.checkActivityFunctions();

console.log(appsThatShouldBeActive); // ['app1']

const appsForACertainRoute = singleSpa.checkActivityFunctions({pathname: '/app2'});

console.log(appsForACertainRoute); // ['app2']

Will call every app's activity function with the `mockWindowLocation` and give you list of which applications should be mounted with that location.

将会调用每个应用的 `mockWindowLocation` 并且返回一个当前location路径字符串，来判断已挂载应用的列表。

###### arguments

mockWindowLocation: string

A string representing a window.location that will be used when calling every application's activity function to test if they return true.

一个代表当前路径的字符串，当执行每个应用的激活函数时会用它来判断是否应该返回真。

###### returns

appNames: string[]

Each string is the name of a registered application that matches the provided `mockWindowLocation`

每个符合当前路径`mockWindowLocation`应该激活的应用名称。

##### 添加错误回调函数

[#](https://single-spa.js.org/docs/api#adderrorhandler)addErrorHandler

singleSpa.addErrorHandler(err => {

console.log(err);

console.log(err.appOrParcelName);

console.log(singleSpa.getAppStatus(err.appOrParcelName));

});

Adds a handler that will be called every time an application throws an error during a lifecycle function or activity function. When there are no error handlers, single-spa throws the error to the window.

添加处理程序，该处理程序将在应用在生命周期函数或激活函数期间每次抛出错误时调用。当没有错误处理程序时，single-spa将错误抛出到window。

###### arguments

errorHandler: Function(error: Error)

Must be a function. Will be called with an [Error object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error) that additionally has a `message` and `appOrParcelName` property.

必须是一个函数。将会以 [Error object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Error) `message` 和 `appOrParcelName`为参数调用.

###### returns

`undefined`

##### 移除错误回调函数

[#](https://single-spa.js.org/docs/api#removeerrorhandler)removeErrorHandler

singleSpa.addErrorHandler(handleErr)

singleSpa.removeErrorHandler(handleErr)

function handleErr(err) {

console.log(err)

}

Removes the given error handler function.

删除给定的错误处理程序函数。

###### arguments

errorHandler: Function

Reference to the error handling function.

参考上边错误处理函数。

###### returns

boolean

`true` if the error handler was removed, and `false` if it was not.

如果错误处理程序被删除，则为“true”;如果没有删除，则为“false”。

##### 挂载根沙箱

[#](https://single-spa.js.org/docs/api#mountrootparcel)mountRootParcel

// Synchronous mounting

// 同步挂载

const parcel = singleSpa.mountRootParcel(parcelConfig, {

prop1: 'value1',

domElement: document.getElementById('a-div')

});

parcel.mountPromise.then(() => {

console.log('finished mounting the parcel!')

})

// Asynchronous mounting. Feel free to use webpack code splits or SystemJS dynamic loading

// 可以随意使用webpack代码分割或SystemJS动态加载

const parcel2 = singleSpa.mountRootParcel(

() => import('./some-parcel.js'),

{ prop1: 'value1',

domElement: document.getElementById('a-div')

});

Will create and mount a [single-spa parcel](https://single-spa.js.org/docs/parcels-overview).

Note: parcels do not automatically unmount. Unmounting will need to be triggered manually.

将会创建并挂载一个 [single-spa 沙箱](https://zh-hans.single-spa.js.org/docs/parcels-overview).

注意:Parcel不会自动卸载。卸载需要手动触发。

###### arguments

parcelConfig: Object or Loading Function

[parcelConfig](/docs/parcels-api#parcel-configuration)

parcelProps: Object with a domElement property

[parcelProps](/docs/parcels-api#parcel-props)

parcelConfig: 对象或加载函数

[沙箱配置](/docs/parcels-api#parcel-configuration)

parcelProps: 对象或一个dom元素 属性

[沙箱属性](/docs/parcels-api#parcel-props)

###### returns

Parcel object

See [Parcels API](https://single-spa.js.org/docs/parcels-api.html) for more detail.

沙箱对象

详细查看[沙箱API](https://single-spa.js.org/docs/parcels-api.html)

##### 支持jquery函数

[#](https://single-spa.js.org/docs/api#ensurejquerysupport)ensureJQuerySupport

singleSpa.ensureJQuerySupport(jQuery);

jQuery uses [event delegation](https://learn.jquery.com/events/event-delegation/) so single-spa must monkey-patch each version of jQuery on the page. single-spa will attempt to do this automatically by looking for `window.jQuery` or `window.$`. Use this explicit method if multiple versions are included on your page or if jQuery is bound to a different global variable.

jQuery使用 [event delegation](https://learn.jquery.com/events/event-delegation/) 所以 single-spa 必须给每个jQuery版本一个patch。single-spa 会试着自动寻找 `window.jQuery` or `window.$`。 如果页面中有多个版本的jQuery存在或jQuery被绑定到多个全局变量，请调用这个的方法。

###### arguments

jQuery?: JQueryFn = window.jQuery

A reference to the global variable that jQuery has been bound to.

对jQuery已绑定到的全局变量的引用。

###### returns

`undefined`

##### 设置初始化最大时长

[#](https://single-spa.js.org/docs/api#setbootstrapmaxtime)setBootstrapMaxTime

// After three seconds, show a console warning while continuing to wait.

// 三秒钟后，在继续等待的同时显示控制台警告。

singleSpa.setBootstrapMaxTime(3000);

// After three seconds, move the application to SKIP\_BECAUSE\_BROKEN status.

// 三秒钟后，将应用程序移动到skip\_because ause\_broken状态。

singleSpa.setBootstrapMaxTime(3000, true);

// don't do a console warning for slow lifecycles until 10 seconds have elapsed

// 在10秒之前不要对缓慢的生命周期发出控制台警告

singleSpa.setBootstrapMaxTime(3000, true, 10000);

Sets the global configuration for bootstrap timeouts.

全局配置初始化超时时间。

###### arguments

millis: number

Number of milliseconds to wait for bootstrap to complete before timing out.

一个判断等待初始化是否超时的毫秒数

dieOnTimeout: boolean = false

If false, registered applications that are slowing things down will cause nothing more than some warnings in the console up until `millis` is reached.

If true, registered applications that are slowing things down will be siloed into a SKIP\_BECAUSE\_BROKEN status where they will never again be given the chance to break everything.

Each registered application can override this behavior for itself.

如果设置为false，注册的应用运行变缓，在到达`millis`前，只会在控制台中引起一些警告。

如果设置为true, 注册的应用程序运行变缓，它们将被塞进一个skip\_break\_status，因为它们不会再打断程序。

每个已注册的应用程序都可以覆盖自己的此行为。

warningMillis: number = 1000

Number of milliseconds to wait between console warnings that occur before the final timeout.

一个判断等待控制台warning是否发生的毫秒数。

###### returns

`undefined`

##### 设置最大挂载时长

[#](https://single-spa.js.org/docs/api#setmountmaxtime)setMountMaxTime

// After three seconds, show a console warning while continuing to wait.

//三秒钟后，在继续等待的同时显示控制台警告。

singleSpa.setMountMaxTime(3000);

// After three seconds, move the application to SKIP\_BECAUSE\_BROKEN status.

//三秒钟后，将应用程序移动到skip\_because ause\_broken状态。

singleSpa.setMountMaxTime(3000, true);

// don't do a console warning for slow lifecycles until 10 seconds have elapsed

//在10秒之前不要对缓慢的生命周期发出控制台警告

singleSpa.setMountMaxTime(3000, true, 10000);

Sets the global configuration for mount timeouts.

全局配置卸载超时时间。

###### arguments

millis: number

Number of milliseconds to wait for mount to complete before timing out.

全局配置挂载超时时间。

dieOnTimeout: boolean = false

If false, registered applications that are slowing things down will cause nothing more than some warnings in the console up until `millis` is reached.

If true, registered applications that are slowing things down will be siloed into a SKIP\_BECAUSE\_BROKEN status where they will never again be given the chance to break everything.

Each registered application can override this behavior for itself.

如果设置为false，注册的应用运行变缓，在到达 `millis`之前，只会在控制台中引起一些警告。

如果设置为true, 注册的应用程序运行变缓，它们将被塞进一个skip\_break\_status，因为它们不会再打断程序。

每个已注册的应用程序都可以覆盖自己的此行为。

warningMillis: number = 1000

Number of milliseconds to wait between console warnings that occur before the final timeout.

一个判断等待控制台warning是否发生的毫秒数。

###### returns

`undefined`

##### 设置卸载最大时长

[#](https://single-spa.js.org/docs/api#setunmountmaxtime)setUnmountMaxTime

// After three seconds, show a console warning while continuing to wait.

//三秒钟后，在继续等待的同时显示控制台警告。

singleSpa.setUnmountMaxTime(3000);

// After three seconds, move the application to SKIP\_BECAUSE\_BROKEN status.

//三秒钟后，将应用程序移动到skip\_because ause\_broken状态。

singleSpa.setUnmountMaxTime(3000, true);

// don't do a console warning for slow lifecycles until 10 seconds have elapsed

//在10秒之前不要对缓慢的生命周期发出控制台警告

singleSpa.setUnmountMaxTime(3000, true, 10000);

Sets the global configuration for unmount timeouts.

全局配置移除超时时间。

###### arguments

millis: number

Number of milliseconds to wait for unmount to complete before timing out.

一个判断等待移除是否超时的毫秒数。

dieOnTimeout: boolean = false

If false, registered applications that are slowing things down will cause nothing more than some warnings in the console up until `millis` is reached.

If true, registered applications that are slowing things down will be siloed into a SKIP\_BECAUSE\_BROKEN status where they will never again be given the chance to break everything.

Each registered application can override this behavior for itself.

如果设置为false，注册的应用运行变缓，在到达`millis`之前，只会在控制台中引起一些警告。如果设置为true, 注册的应用程序运行变缓，它们将被塞进一个skip\_break\_status，因为它们不会再打断程序。

每个已注册的应用程序都可以覆盖自己的此行为。

warningMillis: number = 1000

Number of milliseconds to wait between console warnings that occur before the final timeout.

一个判断等待控制台warning是否发生的毫秒数。

###### returns

`undefined`

##### 设置移除最大时长

[#](https://single-spa.js.org/docs/api#setunloadmaxtime)setUnloadMaxTime

// After three seconds, show a console warning while continuing to wait.

//三秒钟后，在继续等待的同时显示控制台警告。

singleSpa.setUnloadMaxTime(3000);

// After three seconds, move the application to SKIP\_BECAUSE\_BROKEN status.

//三秒钟后，将应用程序切换到skip\_because ause\_broken状态。

singleSpa.setUnloadMaxTime(3000, true);

// don't do a console warning for slow lifecycles until 10 seconds have elapsed

// 在10秒之前不要对缓慢的生命周期发出控制台警告

singleSpa.setUnloadMaxTime(3000, true, 10000);

Sets the global configuration for unload timeouts.

一个判断等待移除是否超时的毫秒数。

###### arguments

millis: number

Number of milliseconds to wait for unload to complete before timing out.

dieOnTimeout: boolean = false

If false, registered applications that are slowing things down will cause nothing more than some warnings in the console up until `millis`is reached.

If true, registered applications that are slowing things down will be siloed into a SKIP\_BECAUSE\_BROKEN status where they will never again be given the chance to break everything.

Each registered application can override this behavior for itself.

如果设置为false，注册的应用运行变缓，在到达`millis`之前，只会在控制台中引起一些警告。如果设置为true, 注册的应用程序运行变缓，它们将被塞进一个skip\_break\_status，因为它们不会再打断程序。

每个已注册的应用程序都可以覆盖自己的此行为

warningMillis: number = 1000

Number of milliseconds to wait between console warnings that occur before the final timeout.

一个判断等待控制台warning是否发生的毫秒数。

###### returns

`undefined`

#### 事件

[#](https://single-spa.js.org/docs/api#events)Events

All of the following are [custom events](https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent/CustomEvent) fired by single-spa on the window. The event `detail` property contains the native DOM event that triggered the reroute, such as a [PopStateEvent](https://developer.mozilla.org/en-US/docs/Web/API/PopStateEvent) or [HashChangeEvent](https://developer.mozilla.org/en-US/docs/Web/API/HashChangeEvent). These events can be handled by using `addEventListener`, like so:

浏览器的所有的下列事件 [custom events](https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent/CustomEvent) 都会被single-spa 触发。

`detail`事件包含触发路由重定向的原生DOM事件，例如 [PopStateEvent](https://developer.mozilla.org/en-US/docs/Web/API/PopStateEvent) 或[HashChangeEvent](https://developer.mozilla.org/en-US/docs/Web/API/HashChangeEvent)。通过 [addEventListener] 可以控制[这些事件](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener), 就像:

window.addEventListener('single-spa:before-routing-event',

evt => {

const originalEvent = evt.detail;

console.log('single-spa event', originalEvent);

})

##### 调用路由前触发事件

[#](https://single-spa.js.org/docs/api#before-routing-event)before routing event

window.addEventListener('single-spa:before-routing-event', () => {

console.log('single-spa is about to mount/unmount applications!');

});

A `single-spa:before-routing-event`event is fired before every routing event occurs, which is after each hashchange, popstate, or triggerAppChange, even if no changes to registered applications were necessary.

每次路由跳转前 `single-spa:before-routing-event`事件会被触发，它可能是 hashchange, popstate, 或者 triggerAppChange，甚至当前应用不需要修改时也会触发。

##### 路由加载触发事件

[#](https://single-spa.js.org/docs/api#routing-event)routing event

window.addEventListener('single-spa:routing-event', () => {

console.log('single-spa finished mounting/unmounting applications!');

});

A `single-spa:routing-event` event is fired every time that a routing event has occurred, which is after each hashchange, popstate, or triggerAppChange, even if no changes to registered applications were necessary; and after single-spa verified that all apps were correctly loaded, bootstrapped, mounted, and unmounted.

每次路由跳转后`single-spa:routing-event`事件会被触发，它可能是 hashchange, popstate, 或者 triggerAppChange，甚至当前应用不需要修改时 ; 在single-spa 校验所有app都正确加载，初始化，挂载，卸载之后此此事件触发。

##### 应用切换时触发事件

[#](https://single-spa.js.org/docs/api#app-change-event)app-change event

window.addEventListener('single-spa:app-change', () => {

console.log('A routing event occurred where at least one application was mounted/unmounted');

});

A `single-spa:app-change` event is fired every time that one or more apps were loaded, bootstrapped, mounted, unmounted, or unloaded. It is similar to `single-spa:routing-event` except that it will not fire unless one or more apps were actually loaded, bootstrapped, mounted, or unmounted. A hashchange, popstate, or triggerAppChange that does not result in one of those changes will not cause the event to be fired.

每次加载，初始化，挂载，卸载或移除一个或多个应用程序时，都会触发 `single-spa:app-change`事件。它与 `single-spa:routing-event`路由事件类似，只是在一个或多个应用程序真正加载，初始化，挂载，卸载或移除之后，它才会启动。如果hashchange、popstate或triggerAppChange不会导致其中任何一个更改，则不会引发事件。

##### 应用未切换时触发事件

[#](https://single-spa.js.org/docs/api#no-app-change-event)no-app-change event

window.addEventListener('single-spa:no-app-change', () => {

console.log('A routing event occurred where zero applications were mounted/unmounted');

});

When no applications were loaded, bootstrapped, mounted, unmounted, or unloaded, single-spa fires a `single-spa:no-app-change` event. This is the inverse of the `single-spa:app-change` event. Only one will be fired for each routing event.

当没有加载，初始化，挂载，卸载或移除应用程序时，single-spa触发 `single-spa:no-app-change`事件。这与 `single-spa:app-change`事件正好相反。每个路由事件只会触发一个

##### 首次挂载前触发事件

[#](https://single-spa.js.org/docs/api#before-first-mount)before-first-mount

window.addEventListener('single-spa:before-first-mount', () => {

console.log('single-spa is about to mount the very first application for the first time');

});

Before the first of any single-spa applications is mounted, single-spa fires a `single-spa:before-first-mount` event; therefore it will only be fired once ever. More specifically, it fires after the application is already loaded but before mounting.

Suggested use case: remove a loader bar that the user is seeing right before the first app will be mounted.

在第一个single-spa应用被挂在之前，single-spa 会触发 `single-spa:before-first-mount`事件；因此它只会触发一次。更具体点说，它只会在应用被加载但未挂载之前触发。

推荐用例： 在用户将要看到第一个应用挂载之前，移除一个loading。

##### 首次挂载触发事件

[#](https://single-spa.js.org/docs/api#first-mount)first-mount

window.addEventListener('single-spa:first-mount', () => {

console.log('single-spa just mounted the very first application');

});

After the first of any single-spa applications is mounted, single-spa fires a `single-spa:first-mount`event; therefore it will only be fired once ever.

Suggested use case: log the time it took before the user sees any of the apps mounted.

在第一个single-spa应用被挂在之后， single-spa 会触发 `single-spa:first-mount` 事件；因此它只会触发一次。

推荐用例： 输出用户看到应用之前花费了多长时间。